FINAL REPORT

MovieSchmmovie: The movie/TV series recommender

Project code:
GitHub repo

MovieSchmmovie is an interactive graph-based IMDb based movie and TV series recommendation system. It is an user friendly console-based program to recommend movies for a certain preference of movie/TV show genre and IMDb. To offer some flexibility to the user, it also offers to choose from the following options:

1. Search for a movie/TV series by it's name
2. Search for the N top movies in a genre
3. Search for the top N TV series for a genre
4. Movie recommendation on the basis of genre and IMDb rating
5. Person search to view their image

MovieSchmmovie is a python-based program that makes use of data scraped from IMDb website and also uses IMDb APIs like IMDbPY (open source) and also API key requiring apis like. The program is based on a graph based algorithm where every node is connected to the other nodes which are related to it, for eg. a movie node with a genre thriller, drama with IMDb rating of 8.2 is connected to the nodes "Thriller", "Drama" and "8.2-8.4".

All the instructions to run the code and required python packages are mentioned in the README.md in the GitHub Repository.

Also, all the interactions with the program are included in the README.md.

The required python packages are as follows:

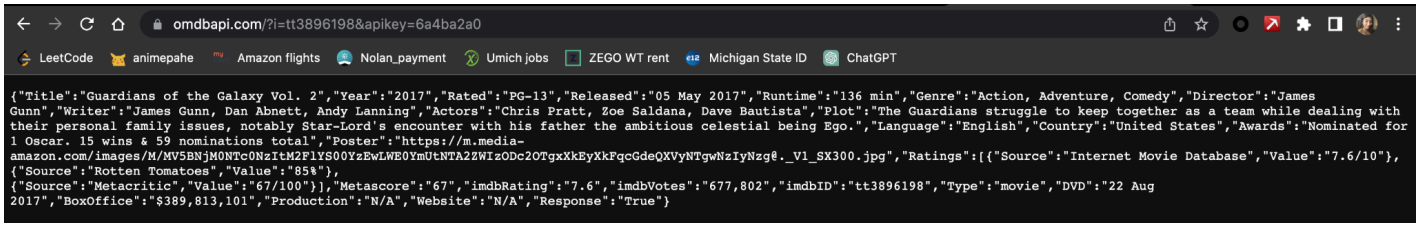
```
python
platform
subprocess
json
imdb
json
imdb
pyglet
sys
os
time
```

```
PIL
urllib.request
bs4
```

Data sources:

The data sources that I used are as follows (*all of the sources are hyperlinked to their urls*):

1. OMDB API
    - OAuth API requires an api key to extract the information and additional verification.
    - I am using the api_key = "6a4ba2a0"
    - The api retrieves the movie metadata in the xml format when the IMDb_movieID is given
    - I used `*requests.get*` in order to retrieve the appropriate metadata from the url containing the information in the **json format**.
    - An example of the information retrieved via the api is shown below:

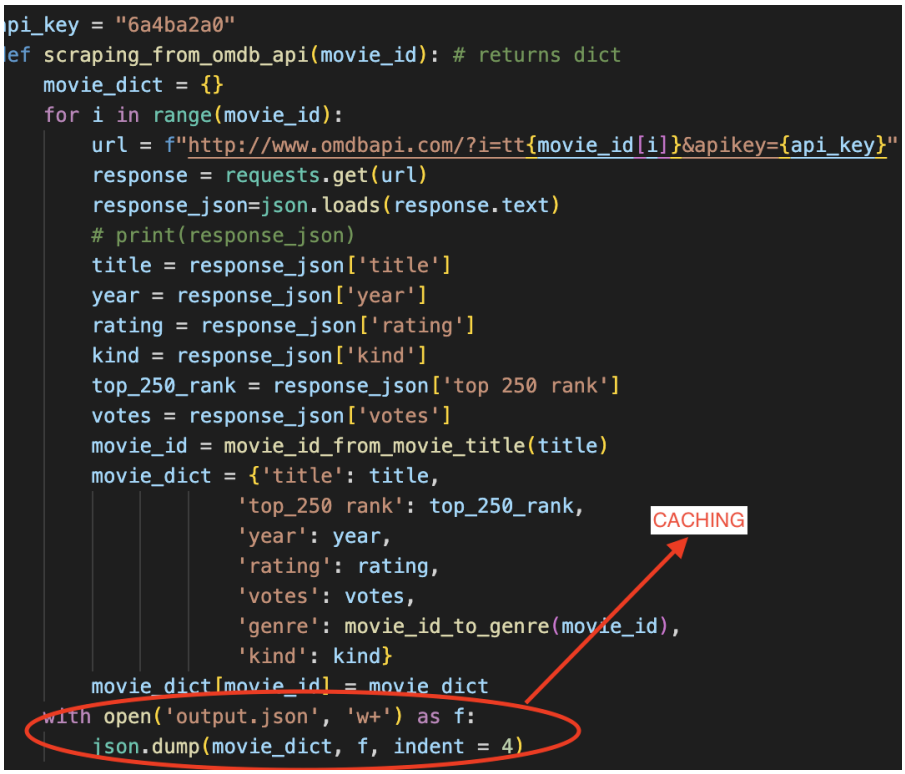


    - Around data for **100** movies was scraped using this api. The attributes of movie retrieved was as follows:
        - Title
        - Year
        - Rated: pg-13, U/A, etc.
        - Release date
        - Runtime
        - Movie Rank in top 250 if it's rank exists in top 250
        - Genre
        - Director
        - Writer
        - Actors
        - Plot
        - Awards
        - imdbRating
        - imdbVotes
        - imdbID
        - type

- I used this data to store the movie details of 100 movies in the **cache** named `*movies_cache.json*` in the json format to be used as nodes for my graph structure for the movie recommendation system. I appended the data from all the 3 sources to get the cache `*movies_cache.json*`.
- Records available ~1M movies' metadata
- Records retrieved: 100 movies' metadata
- **Evidence of caching**:


```
    },
    "0317248": {
        "title": "City of God",
        "top_250 rank": 23,
        "year": 2002,
        "rating": 8.6,
        "votes": 756415,
        "genre": "Crime, Drama",
        "kind": "movie"
    },
    "0120815": {
        "title": "Saving Private Ryan",
        "top_250 rank": 24,
        "year": 1998,
        "rating": 8.6,
        "votes": 1388132,
        "genre": "Drama, War",
        "kind": "movie"
    },
    "0118799": {
        "title": "Life Is Beautiful",
        "top_250 rank": 25,
        "year": 1997,
        "rating": 8.6,
        "votes": 694301,
        "genre": "Comedy, Drama, Romance, War",
        "kind": "movie"
    },
```

- **Part of code showing caching:**



2. IMDBpy API:
    - It is an open source API requiring no
    - Around data for **250** movies was scraped from the IMDbPY api.
    - After installing the IMDBpy module by using *`pip install imdbpy`,* I created an instance of IMDB.
    - Extracted the required movie details in the **XML format**.
    - Parsed the XML script for each movie in order to extract the useful information and to display the details to the user in a readable, understandable format.
    - I extracted various attributes for a movie such as:
        a. Movie Title
        b. Movie year
        c. Movie genres
        d. Movie Rank in top 250 if it's rank exists in top 250
        e. Movie IMDb ratings from 1-10
        f. Movie runtime
        g. Movie IMDb ID
        h. Cast
        i. Director/directors
        j. Kind: movie, documentary, etc.

    k. short_Overview
    l. Summary plot
- I also used this api to extract the IMDb IDs of the movies so that I could use those IMDB IDs as input to the OMDB API to extract the movie information using that API.
- I used this data to store the movie details of 250 movies in the **cache** named `*movies_cache.json*` in the json format to be used as nodes for my graph structure for the movie recommendation system.
- Records available ~1M movies' metadata
- Records retrieved: 250 movies' metadata
- **Evidence of caching**:

- **Part of code showing caching:**

```python
def scraping_from_imdbpy():
    moviesDB = imdb.IMDb()
    movie = moviesDB.get_movie()
    movie_dict = {}
    for i in range(len(movie)):
        title = movie[i]['title']
        year = movie[i]['year']
        rating = movie[i]['rating']
        kind = movie[i]['kind']
        top_250_rank = movie[i]['top 250 rank']
        votes = movie[i]['votes']
        movie_id = movie_id_from_movie_title(title)
        movie_dict = {'title': title,
                      'top_250 rank': top_250_rank,
                      'year': year,
                      'rating': rating,
                      'votes': votes,
                      'genre': movie_id_to_genre(movie_id),
                      'kind': kind}
        movie_dict[movie_id] = movie_dict
    with open('movies_cache.json', 'w+') as f:
        json.dump(movie_dict, f, indent = 4)
```

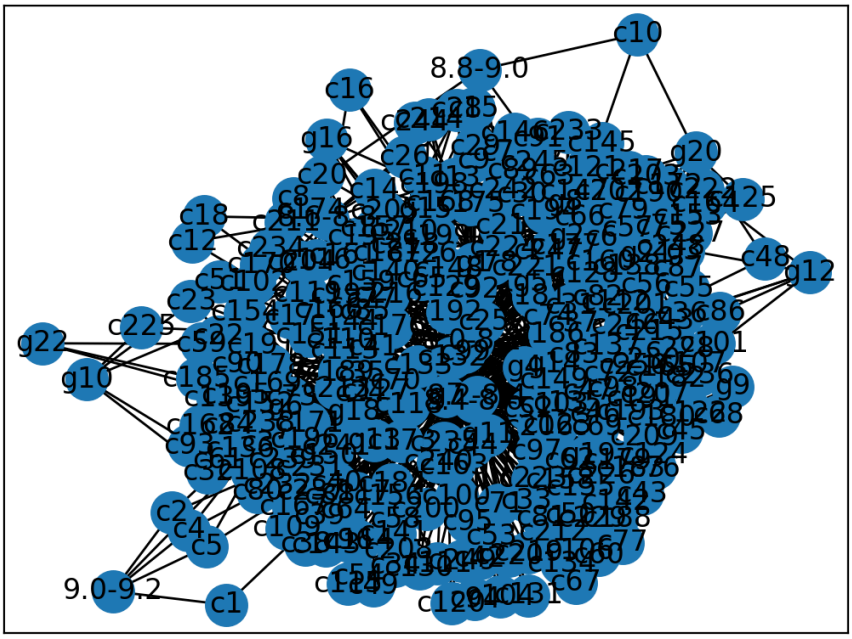
CACHING

3. [IMDb website scraping]
    - I used Beautiful Soup 4 to scrape data from the [IMDb website].
    - Records available ~1M movies' metadata
    - Records retrieved: 100 movies' metadata
    - Around data for **100** movies was scraped using this api. The attributes of movie retrieved was as follows:
        - Title
        - Year
        - Rated: pg-13, U/A, etc.
        - Release date
        - Runtime
        - Movie Rank in top 250 if it's rank exists in top 250
        - Genre
        - Director
        - Writer
        - Actors
        - Plot
        - Awards
        - imdbRating
        - imdbVotes
        - imdbID
        - type

- I used this data to store the movie details of **100** movies in the **cache** named `*movies_cache.json*` in the json format to be used as nodes for my graph structure for the movie recommendation system.

Data Structure:

- The README.md file has the graph data structure explained in brief.
- The entire movies and TV series data is stored in the form of cache in `movies_cache.json`
- A visualization of the nodes connection is as follows:
  (used python script in `**nodes_visualize_adjacency_list.py**` to create this plot)



- A graph structure is used to store this dataset in order to retrieve nodes when the recommendation system is active.
- Python file that constructs the graph: `**graph_structure.py**` (pictures are shown step by step during explanation)
- I have constructed an undirected graph by treating each movie as a separate child node and tokenizing each of the movies as nodes ‘c1’, ‘c2’, ‘c3’, ……An image showing the

creating and storing the child nodes is given below:

```
def create_movie_info_node():
    """
    # returns dict for cache movies in the format: {"1": {"title": ...,
                                                        "top_250 rank": ...,
                                                        "year": ...,
                                                        "rating": ...,
                                                        "votes": ...,
                                                        "genre": ...",
                                                        "kind": ...}, }
    """
    with open('/Users/upasanathakuria/Desktop/507/SI507_final_project/top250movies.json', 'r') as file:
        data = json.load(file)

    movie_node_info = {}
    for i, (_, value) in enumerate(data.items()):
        new_key = 'c' + str(i + 1)
        movie_node_info[new_key] = value
    return movie_node_info
```

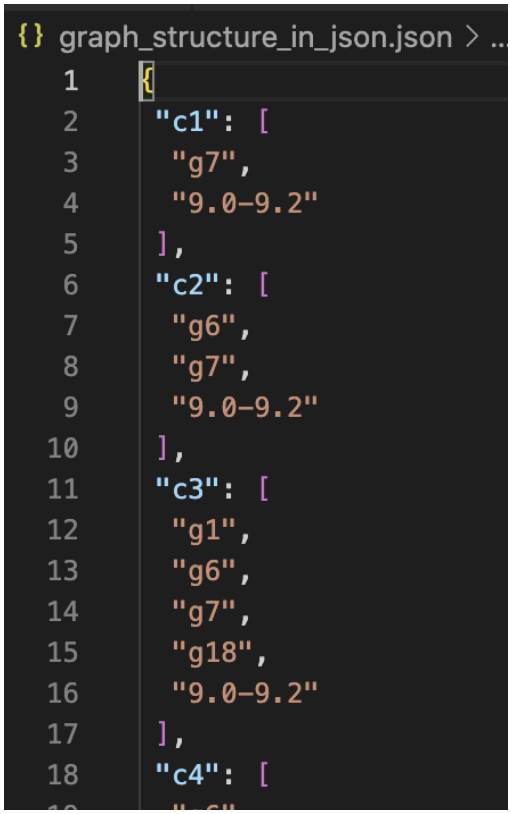
- Each of the movie nodes is connected to other nodes and stored in the form of a graph. For example, each if a movie named “Lol” has genres as comedy, thriller, romance and imdb of 8.4 in the cached dataset, then node “Lol” in the form of ‘c#’ is mapped to the nodes representing genres comedy, thriller, romance in the form of ‘g#’ and also connected to the node “8.4-8.6” rating node.
  Mapping example:
  {“c#”: [g#, g#, g#, “8.4-8.6”]
  …}
  The code snippet showing the movie nodes connection through an adjacency list to the other nodes is shown below (data taken from the cached data info):

```
def create_adjacency_dict_child_nodes():
    data_dict = create_movie_info_node()
    adjacency_dict = {}
    inverted_genre_rating_info_dict = invert_keys_values_dict(create_genre_rating_info_node(

    for i, data in data_dict.items():
        each_item_list = []
        genres_each_movie = (data['genre'].split(","))
        genres_each_movie = [i.lstrip() for i in genres_each_movie]
        for each_genre in genres_each_movie:
            each_item_list.append(inverted_genre_rating_info_dict[each_genre])
        # print(each_item_list)
        if data['rating'] < 8.2:
            each_item_list.append(rating_list[0])
        elif data['rating'] < 8.4:
            each_item_list.append(rating_list[1])
        elif data['rating'] < 8.6:
            each_item_list.append(rating_list[2])
        elif data['rating'] < 8.8:
            each_item_list.append(rating_list[3])
        elif data['rating'] < 9.0:
            each_item_list.append(rating_list[4])
        elif data['rating'] <= 9.2:
            each_item_list.append(rating_list[5])
        adjacency_dict[i] = each_item_list
    return adjacency_dict
```

- Every genre and rating is tokenized as parent nodes. Each genre is tokenized as '**g1**', '**g2**', '**g3**', ….. And each rating is tokenized with respect to it's IMDb rating range **["8.0-8.2", "8.2-8.4", "8.4-8.6", "8.6-8.8", "8.8-9.0", "9.0-9.2"]**
- Just like each child movie node is mapped to each of it's genres and rating, each individual genre is also connected to it's related child node. Same way each of the individual rating is mapped to it's related child node.
- Hence each of the movie nodes are connected to it's genre nodes and imdb rating window nodes and vice versa.
- The implementation of the graph structure in the code that shows the connections of the each node is shown below:

```python
def graph_implementation():
    idx = 1
    for i in genre_list:
        print(f"{idx}) {i}")
        idx += 1
    input_genres = input(f"Enter the integers for the genres you want in the format of space separted intergers like 1 2 3: ")
    input_genres = input_genres.split(" ")
    # print(input_genres)
    genre = [genre_list[int(i)-1] for i in input_genres]
    idx_r = 1
    for i in rating_list:
        print(f"{idx_r}) {i}")
        idx_r += 1
    input_rating = input(f"Enter the integers for the IMDb rating range you want in the format of space separted intergers like 1 2 3: ")
    input_rating = input_rating.split(" ")
    rating = [rating_list[int(i)-1] for i in input_rating]

    inverse_genre_rating_node = invert_keys_values_dict(create_genre_rating_info_node())

    adj_list = create_adjacency_dict_child_nodes()
    active_list = []

    for gen in genre:
        active_list.append(inverse_genre_rating_node[gen])

    for rat in rating:
        active_list.append(inverse_genre_rating_node[rat])

    # active list = ["g1", "g2", .... ,"r1", "r2", ....]
    results_rating, results_genre = [],[]
    for idx, (key, val) in enumerate(adj_list.items()):
        if(key[0]!="c"):
            continue
        for vav in active_list:
            if(vav in val and vav[0]=="g"):
                results_genre.append(key)
            else:
                results_rating.append(key)
```

- The python file showing the graph structure is named `**graph_structure_in_json.json**` and a snippet is shown below:



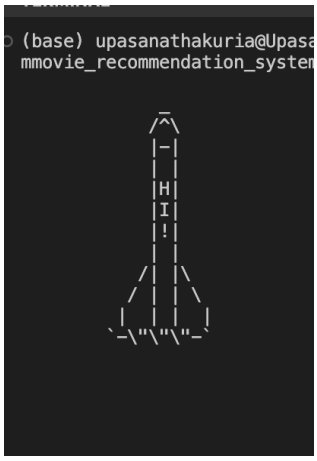
- **Standalone python file that reads the graph structure in json `reading_graph_from_json`:**

```python
import json

def create_dict_from_json():
    with open('./graph_structure_in_json.json', 'r') as file:
        data = json.load(file)
    print(data)
    return data

create_dict_from_json()

"""
Returns
{'c1': ['g7', '9.0-9.2'], 'c2': ['g6', 'g7', '9.0-9.2'], 'c3'
"""
```

Interaction and presentation: (interaction also mention in README.md)

For interaction purposes, I've implemented the following in the terminal in the form of animations. The following is the step-by-step walkthrough:

1. The user is introduced to the program in the starting by **a rocket that travels up in motion** greeting "HI":



2. Next, the user is asked their name, and the program takes the name as the input.
3. The user is displayed with an animation of **moving smoke out of a house** with the display:

4. Next I have animated to get dancing characters to greet the user shown in the image below:

```
hELLO cHRISTOPHER nOLAN lET ME GET YOU Started 😆\
```

5. Next it displays the options shown below:

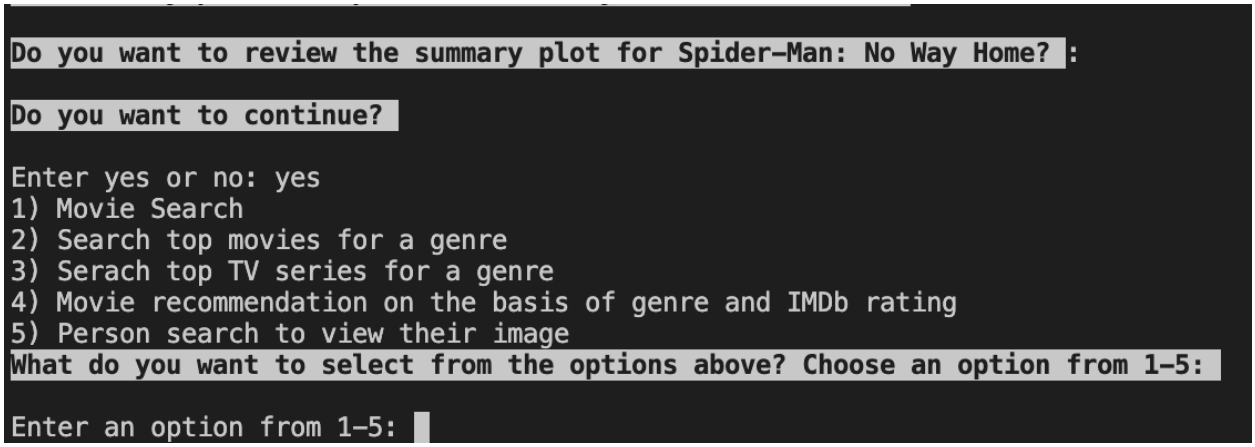
```
1) Movie Search
2) Search top movies for a genre
3) Serach top TV series for a genre
4) Movie recommendation on the basis of genre and IMDb rating
5) Person search to view their image
What do you want to select from the options above? Choose an option from 1-5:

Enter an option from 1-5:
```

6. Next it asks the user what they would like to search for and takes the user input, in this case my input was "**spiderman**". It displays the top results for that word and asking which one to select to display results for:

```
1) Movie Search
2) Search top movies for a genre
3) Serach top TV series for a genre
4) Movie recommendation on the basis of genre and IMDb rating
5) Person search to view their image
What do you want to select from the options above? Choose an option from 1-5:

Enter an option from 1-5: 1
What is the movie/tv series name you want to search?

Enter a name for the movie/tv search: spiderman
0: Spider-Man
1: Spider-Man: No Way Home
2: Spiderman the Verse
3: The Amazing Spider-Man
4: Spider-Man: Into the Spider-Verse
5: Spider-Man: Homecoming
6: The Amazing Spider-Man 2
7: Italian Spiderman
8: Spider-Man 3
9: Spider-Man: Far from Home
10: Spider-Man: The Animated Series
11: Spider-Man 2
12: Miles Morales Ultimate Spiderman
13: The Amazing Spider-Man
14: Spider-Man: Across the Spider-Verse
15: Spider-Man
16: The Amazing Spiderman 2 Webb Cut
17: Spiderman 5
18: Spider-Man
19: Spider-Man
Which movie's details you want to look for?
```

7. Next, user can select any number from 0-19 to look out for details of the movies. A demo is shown below:

8. Next it displays the plot if it is available too. On ending the search, it asks the user if they want to continue with the program or not:



9. We can again select an option to avail the rest of the options as seen above.
10. The **option 4: Movie recommendation** makes use of the graph structure that stores the cached data. When option 4 is selected it asks the user the following:

```
5) Person search to view their image
What do you want to select from the options above? Choose an option from 1-5:

Enter an option from 1-5: 4
1) Action
2) Adventure
3) Animation
4) Biography
5) Comedy
6) Crime
7) Drama
8) Fantasy
9) History
10) Horror
11) Music
12) Musical
13) Mystery
14) Romance
15) Sci-Fi
16) Sport
17) Superhero
18) Thriller
19) War
20) Western
21) Family
22) Film-Noir
Choose the integers for the genres you want in the format of space separted intergers like 1 2 3.

Enter those integers in the format of space separted intergers like 1 2 3: 1 2
1) 8.0-8.2
2) 8.2-8.4
3) 8.4-8.6
4) 8.6-8.8
5) 8.8-9.0
6) 9.0-9.2
Choose the integers for the IMDb rating range you want in the format of space separted intergers like 1 2 3.

Enter the integers in the format of space separted intergers like 1 2 3: 1
```

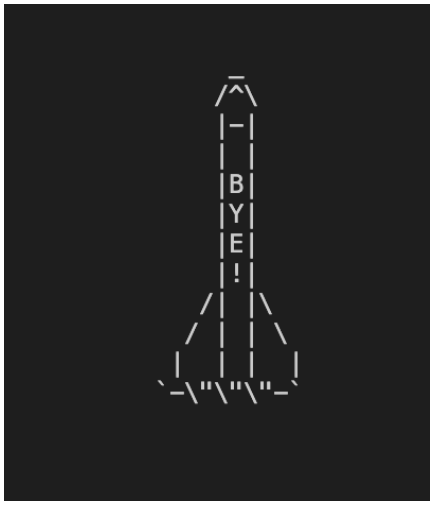
11. When the user inputs the genres and IMDB rating in the format: “1 2 3” meaning Action Adventure and animation genres, the program recommends the user howsoever number of movies that satisfies those inputs:

```
Choose the integers for the IMDb rating range you want in the format of space separted intergers like 1 2 3.

Enter the integers in the format of space separted intergers like 1 2 3: 1
How many movie recommendations do you want?

Enter an integer 1-86. If you want the full recommendation list, enter 'full'. To exit enter 'exit': 30
1) Ran
2) Die Hard
3) Batman Begins
4) Spirited Away
5) The Dark Knight
6) Ford v Ferrari
7) The Treasure of the Sierra Madre
8) Lawrence of Arabia
9) Inception
10) Rush
11) Lock, Stock and Two Smoking Barrels
12) WALL·E
13) Indiana Jones and the Last Crusade
14) Coco
15) Pirates of the Caribbean: The Curse of the Black Pearl
16) Logan
17) The Matrix
18) Gladiator
19) Inglourious Basterds
20) Barry Lyndon
21) The Dark Knight Rises
22) Spider-Man: No Way Home
23) The Bridge on the River Kwai
24) Blade Runner
25) Warrior
26) Seven Samurai
27) Avengers: Endgame
28) Ratatouille
29) 2001: A Space Odyssey
30) Monsters, Inc.
Do you want to continue?

Enter yes or no:
```

12. If user wants to continue they can type ‘yes’ and ‘no’ to exit, the bye animations are as follows:

```
Enter yes or no: no
sEE YOU LATER cHRISTOPHER nOLANL HOPE YOU HAD A GOOD TIME 😃 See you next time 👋
```

A BYE rocket that travels up when exit



[Demo Link]

https://drive.google.com/file/d/1Q7EEZ73ioP1CTyWMHOcbtb4DmZhpanDi/view?usp=share_link